UM EECS 270 F22
Introduction to Logic Design

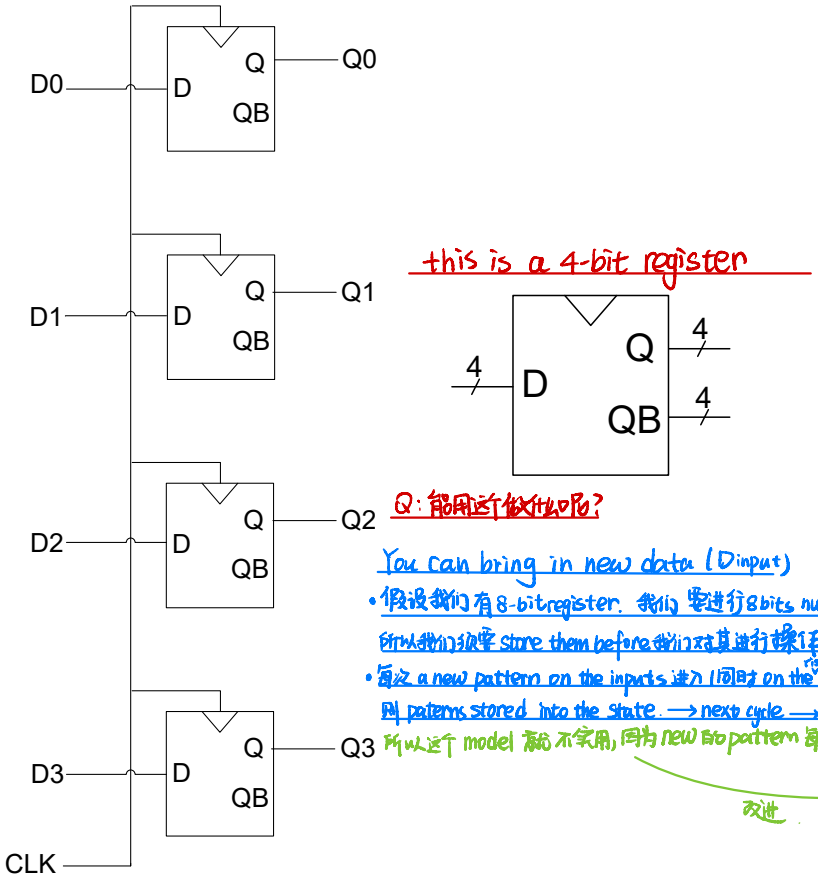
14. Sequential Building Blocks

Sequential Blocks

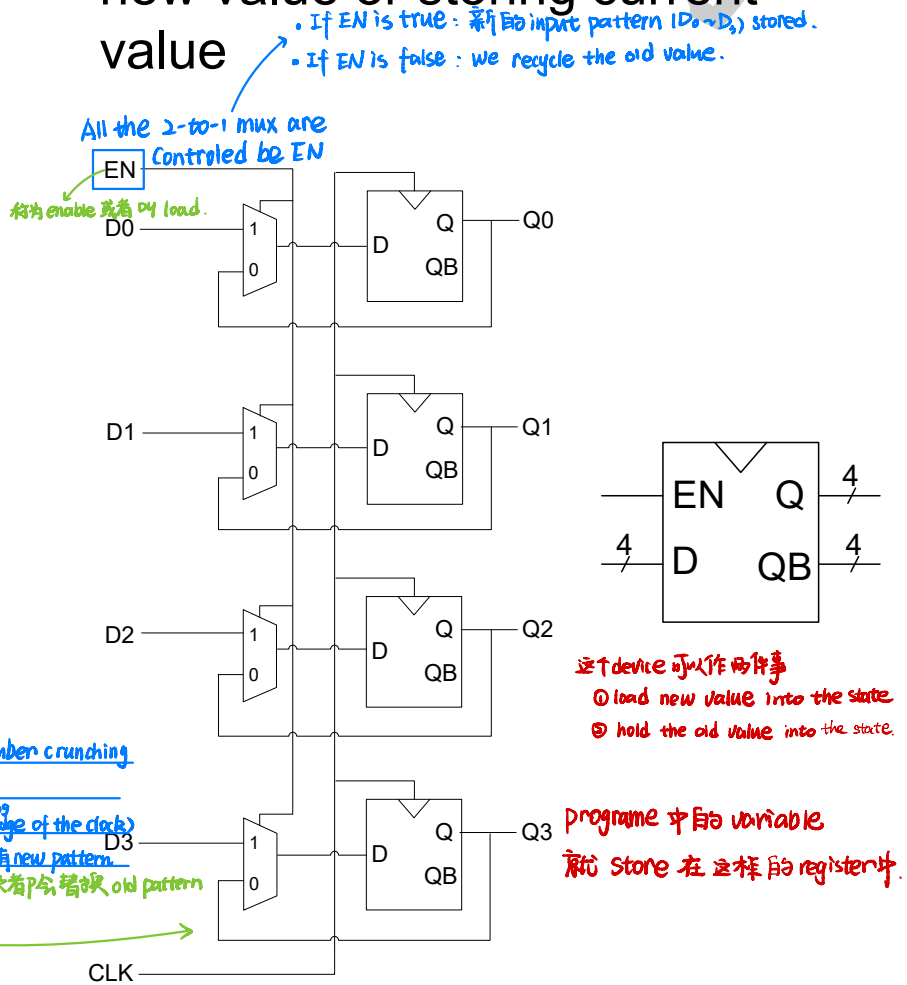
- Goal is to create a toolbox of frequently used sequential devices
  - Registers
  - Counters
    - Ripple counter
    - Parallel counter
  - Shift Registers
    - Parallel in shift register
    - Universal shift register
  - Shift Register Counters
    - Ring counter
    - Johnson counter
    - Linear feedback shift register

Registers

- A collection of two or more D flip-flops with a common clock is called a **register**
- Used to store a collection of bits

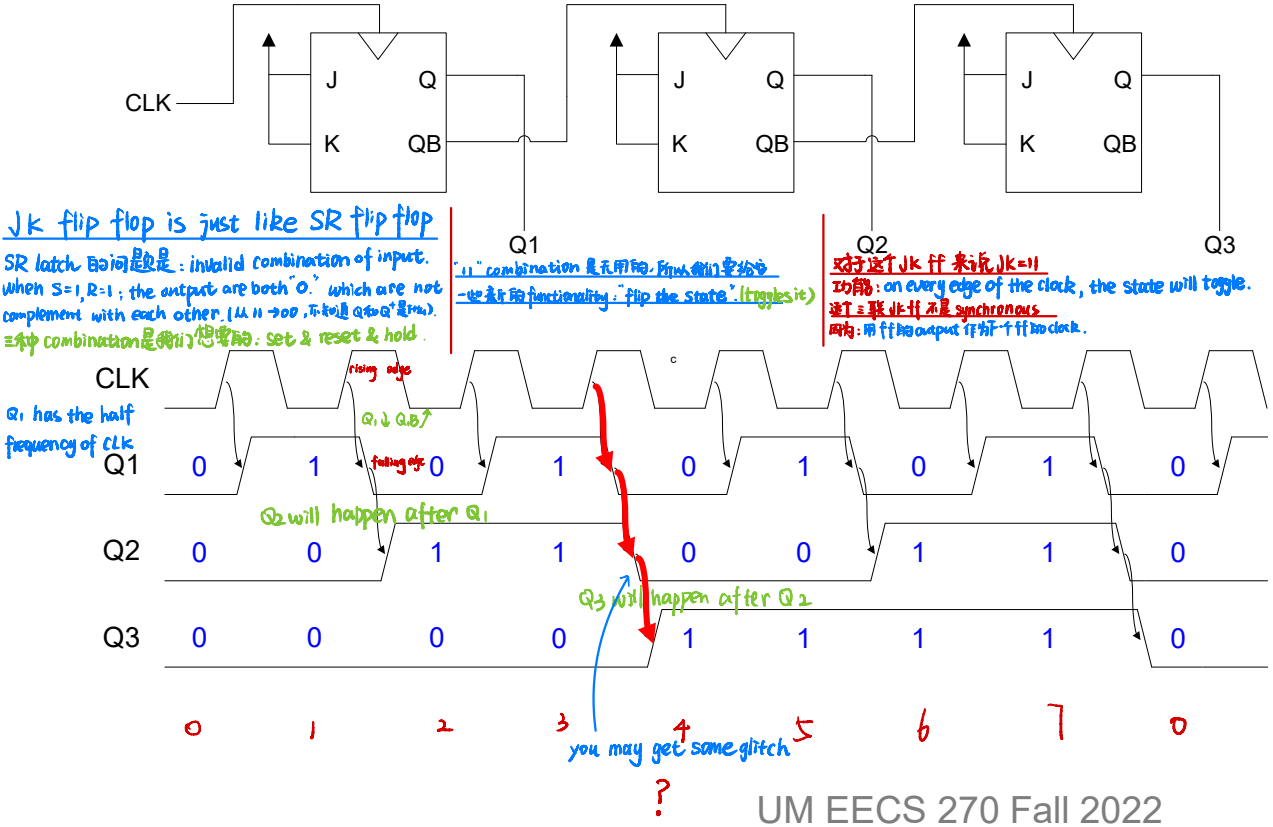


- Adding a mux at the input of each D FF allows for loading new value or storing current value



Counters

- Very common sequential circuit
- No physical counter can count forever – real counters are **modulo counters**
  - I.e., they "count" through a finite number of states and then repeat
  - The number of states in a counter's sequence is called the **modulus** of the counter
  - Example: A counter that counts the sequence 0, 1, 2, …, 7, 0, 1,… is a *modulo 8 counter*
- A **ripple counter** is an easy counter implementation



What is the modulus of this counter?

8

- State changes *ripple* through the circuit
- Outputs don't change at the same time!

solution is to go synchronous

what we need : All the flip flops be controled by the same clock

所以我们不能光有 flip flop，我们还得有其他 logic.

Parallel Counter

- Would like *all* state variables to run on the same clock signal
- Examining the binary code yields an easy implementation:

the 1sth bit toggles on the primary clock
the 2nd bit toggles on the complement of the 1st bit
the 3rd bit toggles on the complement of the 2nd bit

b = Q

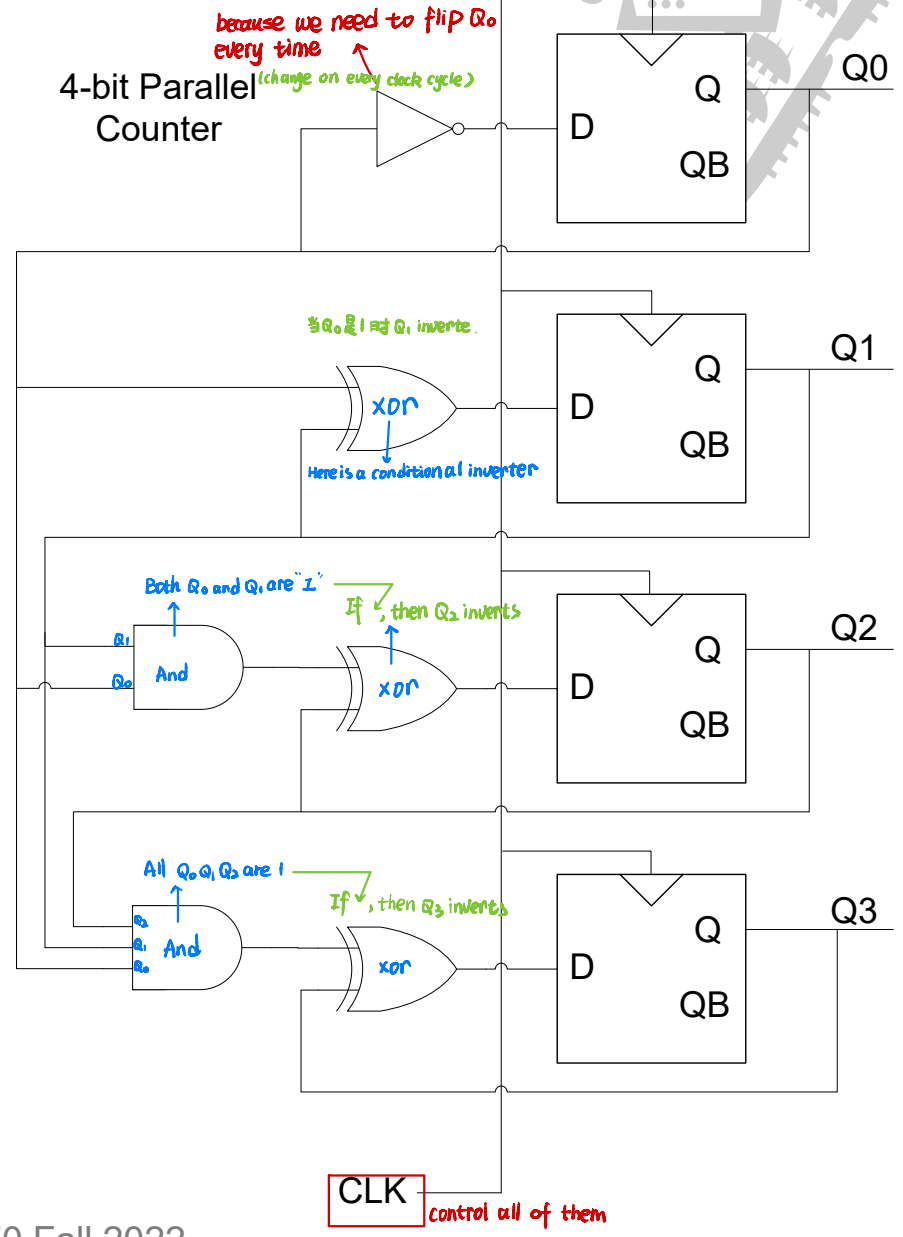
b_2	b_1	b_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0

当 b_0 是 1 时，b_1 要 toggle

when the less significant bits (b_0 & b_1) become max, (they are all one)

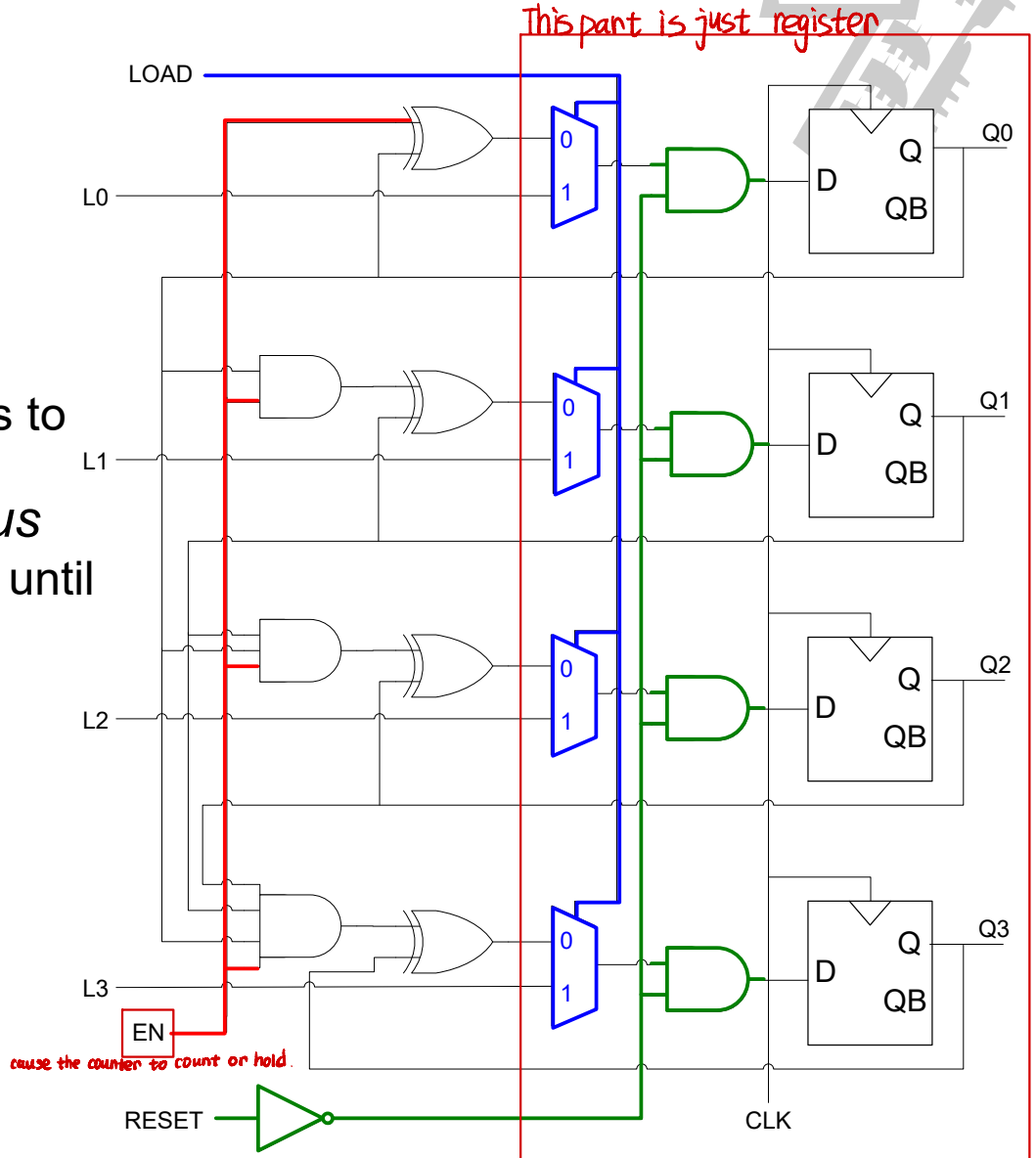
when flip a bit? (除) b_0)
when all bits before it are all one.

- Under what condition does each bit toggle?
- *When all bits of lesser significance are 1!*
- This is true no matter how many bits in the counter



Parallel Counter with Enable, Load, and Reset

- New counter inputs
  - **RESET**: reset all state variables to 0
  - **ENABLE**: enable = 1 → count, enable = 0 → hold
  - **LOAD**: set state variables to load input values
- New inputs are *synchronous*
  - Their effects are not seen until the clock edge



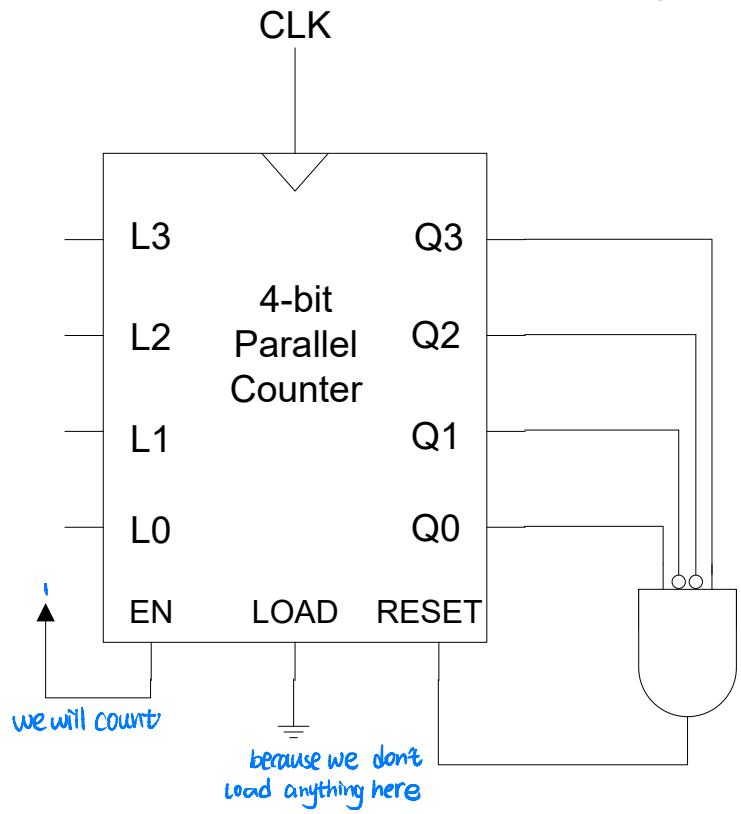
Using a Modulo *n* Counter to Make a Modulo *m* Counter (*m* < *n*)

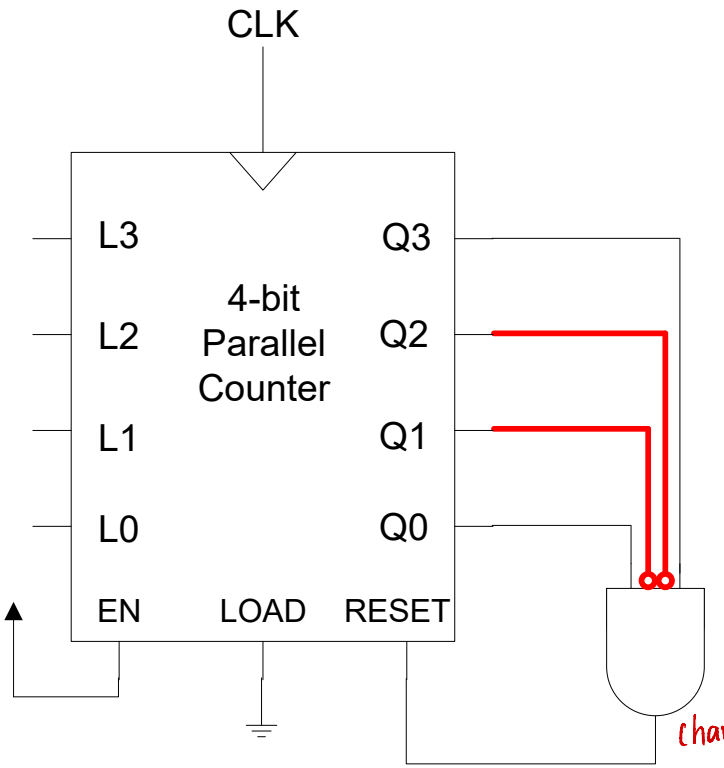
- Want to implement a modulo 10 counter with the sequence:

  0, 1, 2, …, 9, 0, 1,..

  without designing a new counter from scratch

- Use modulo 16 counter with a few modifications
  - Need to reset the counter during the counting sequence
  - When should RESET be asserted?
    - When state 9 (1001) is detected on the state variables!





9 is the first state where both Q3 and Q0 equal 1

change 4 input and gate to 2 input and gate

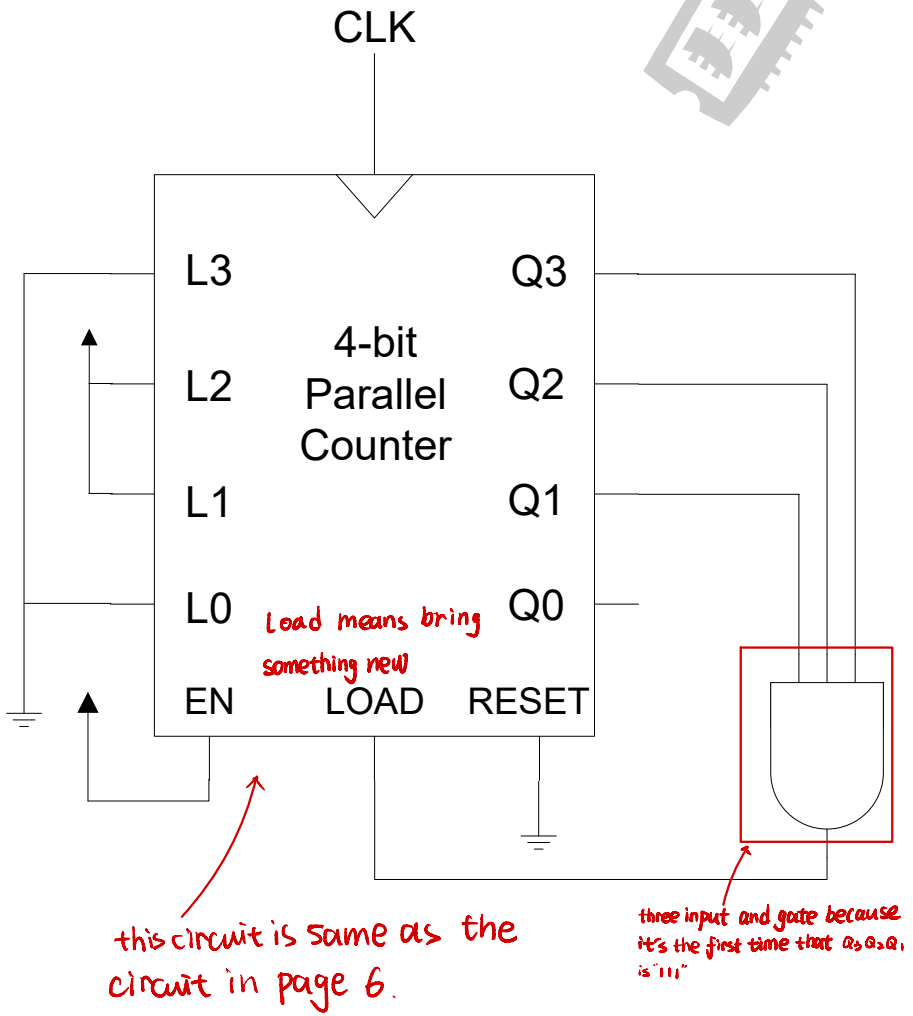
Q3	Q2	Q1	Q0	#
0	0	0	0	*0*
0	0	0	1	*1*
0	0	1	0	*2*
0	0	1	1	*3*
0	1	0	0	*4*
0	1	0	1	*5*
0	1	1	0	*6*
0	1	1	1	*7*
1	0	0	0	*8*
1	0	0	1	*9*
1	0	1	0	*10*
1	0	1	1	*11*
1	1	0	0	*12*
1	1	0	1	*13*
1	1	1	0	*14*
1	1	1	1	*15*

Are these AND gate inputs necessary??

New goal : count from 6 to 14. (How to count arbitrary sequences).
we have to load 6, and then when we get to 14. we should go back and load 6.

- Implement a modulo 9 counter with the counting sequence

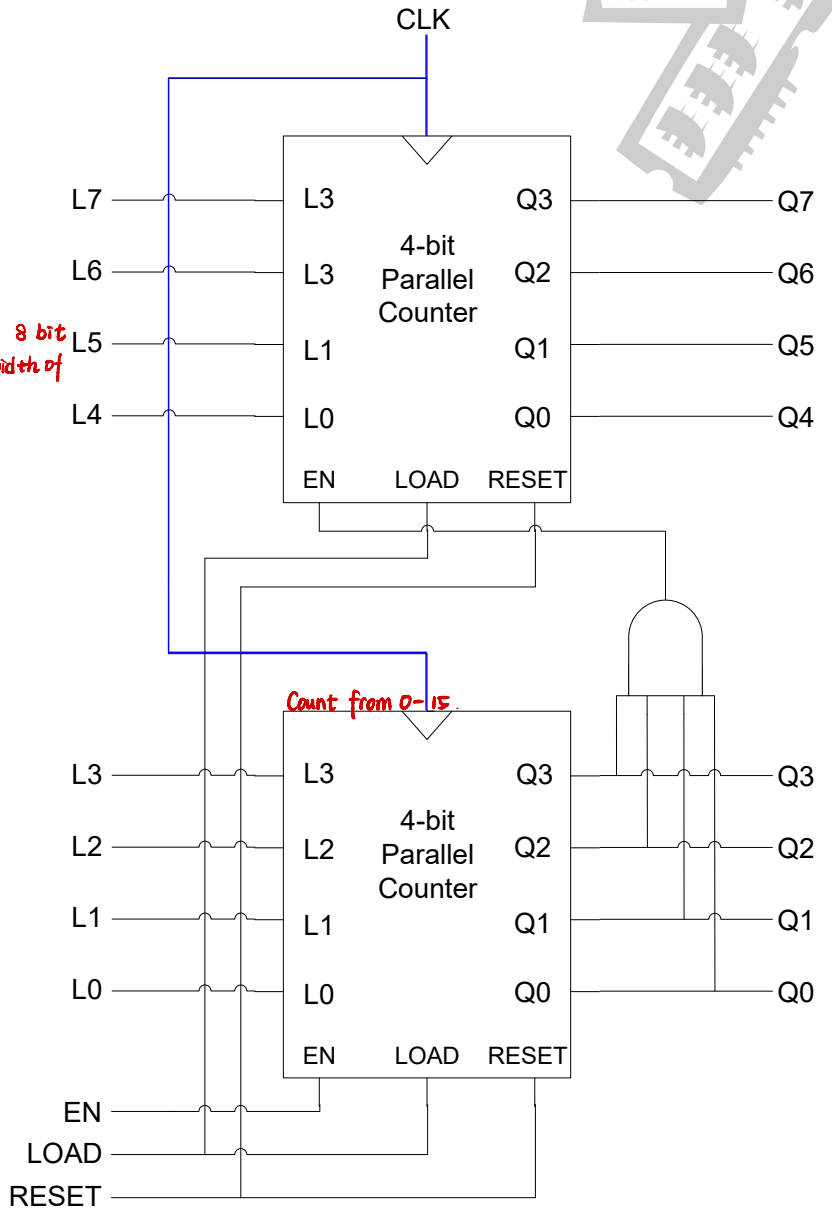
  6,7,…,13,14,6,7,…
- What value to load?
  - 6 (0110)
- When to load?
  - when 14 (1110) is detected on outputs



Cascading Counters

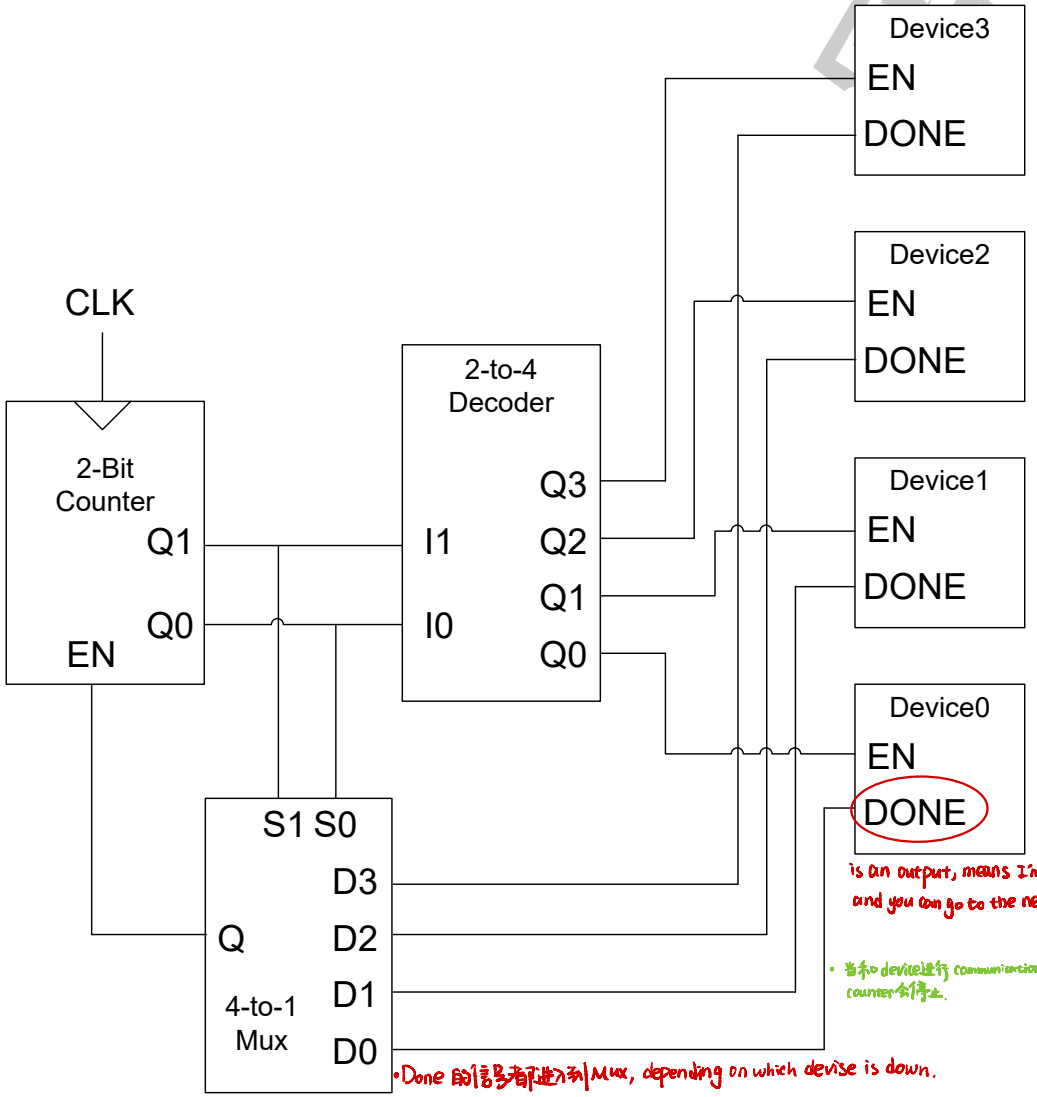
- Buliding an 8-bit binary counter from two 4-bit binary counters…
  - Both counters are clocked by the same signal
  - When should the higher order counter be enabled?
    - When all bits of the lower order counter are 1!

So that it can perform the 8 bit counting without increasing the width of the gates.



Counter Application: Polling

- Combine counter with decoder to create a 1-out-of-n sequence
- **Polling**: Cycle through devices, enabling each in turn



存在的问题：

如果device3失联了，则未运行会raise done.

则整个系统stuck. (hangs)

改进方法：

给予另一个条件：时间.

Give you 10 seconds, if you don't give me feedback, then move on to the next device.

is an output, means I'm done and you can go to the next device

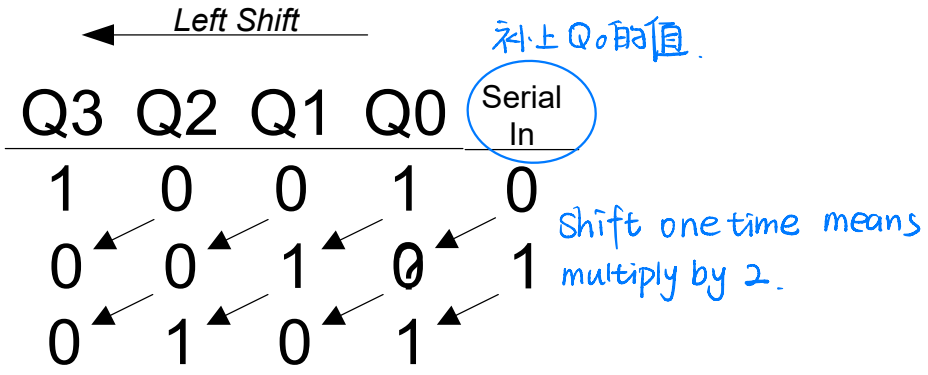
• 当和device进行 communication时 counter会停止.

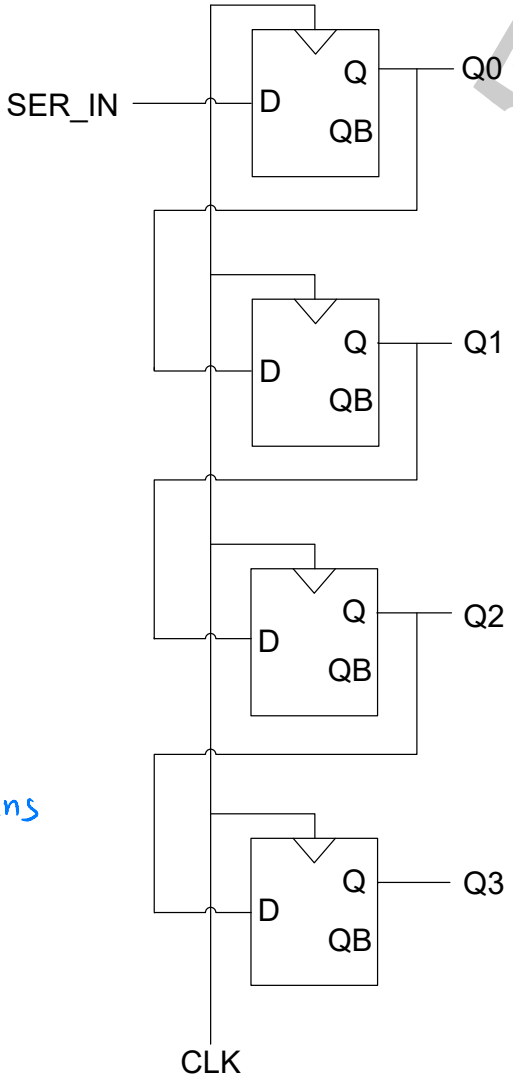
• Done的信号都进入到Mux, depending on which device is down. 来决定接下来和哪个device communicate

• the done signal is false for all of them initially. So you need to wait until the done signal for the device becomes true, which would enable the counter to continue counting.

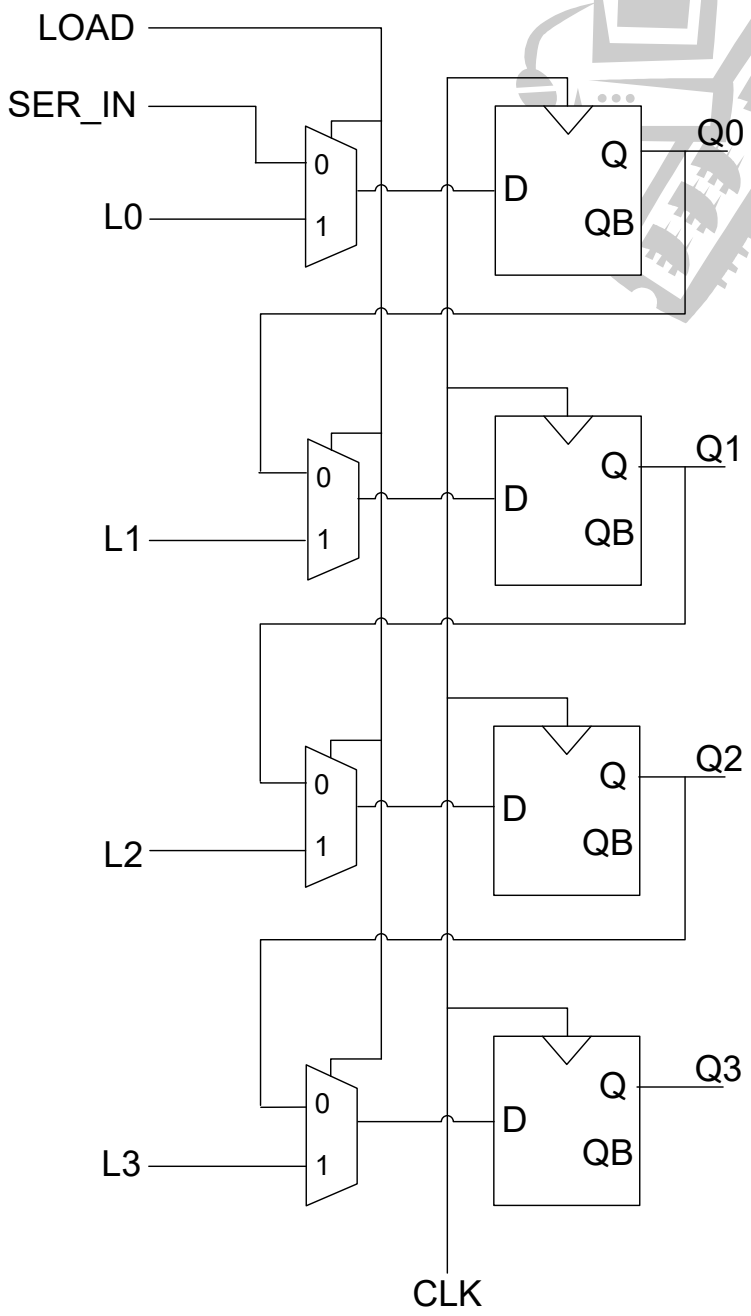
Shift Registers

- A **shift register** allows for the shifting of its bits by one bit position on every clock edge

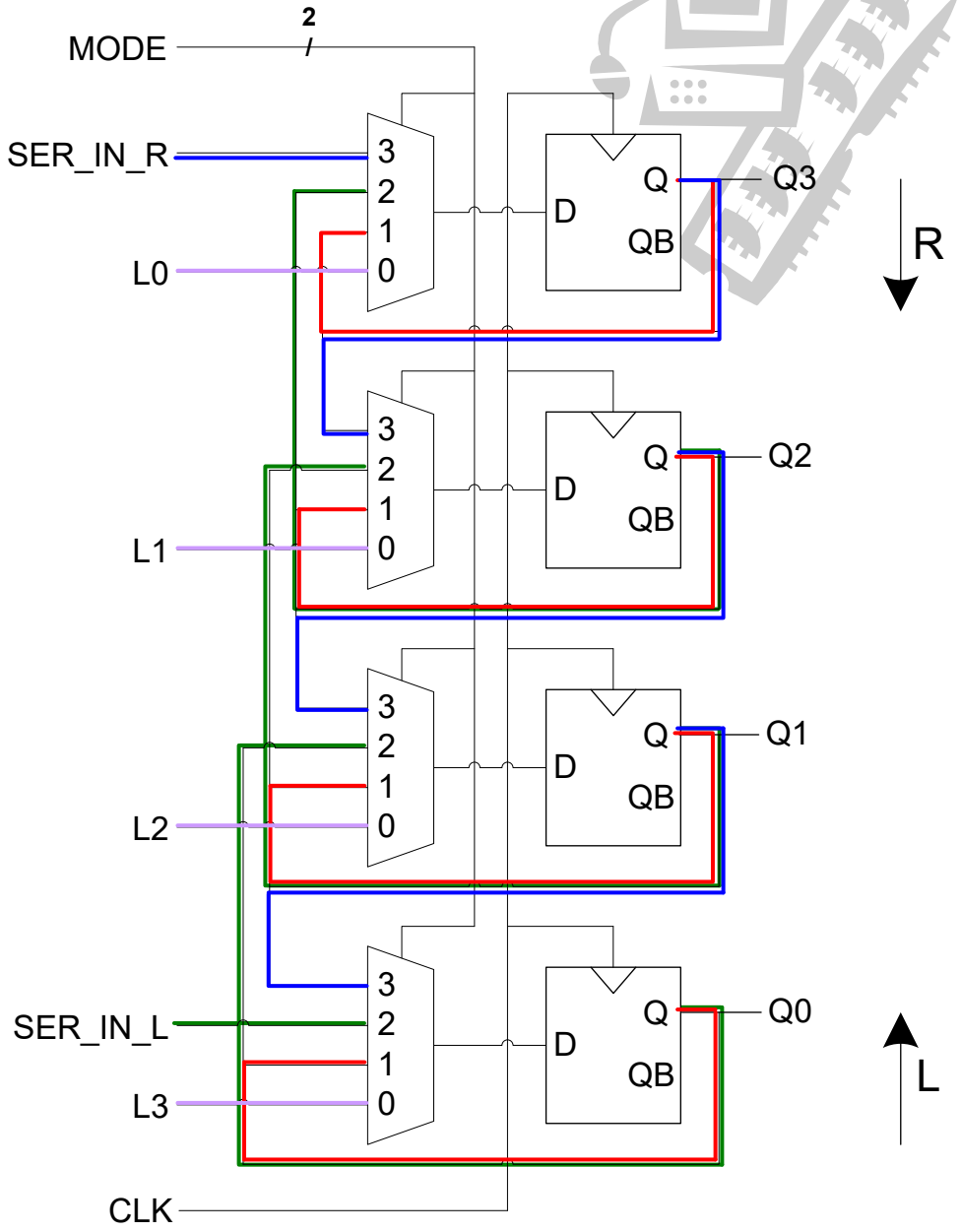




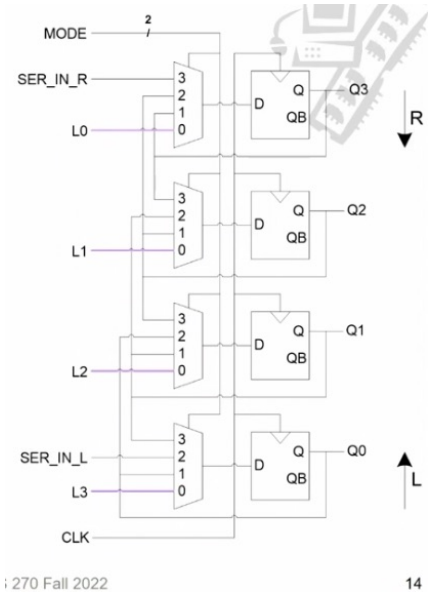
- A **parallel-in** shift register has additional inputs that allow for the loading of all register bits on a single clock edge
- Implementation
  - Use a mux to select between shifting mode (LOAD = 0) and loading mode (LOAD =1)



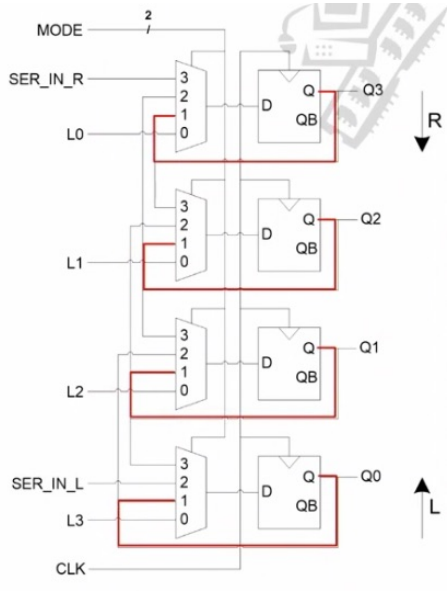
- A **universal shift register** has four modes of operation
  - Parallel load
    - MODE = 0 (00)
  - Hold
    - MODE = 1 (01)
  - Shift Left
    - MODE = 2 (10)
  - Shift Right
    - MODE = 3 (11)



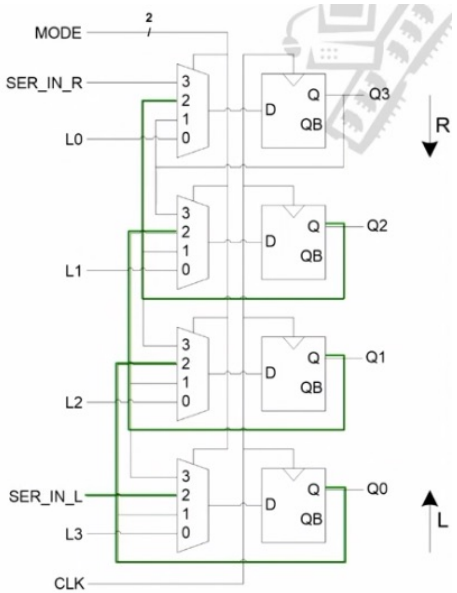
– Parallel load
  - MODE = 0 (00)



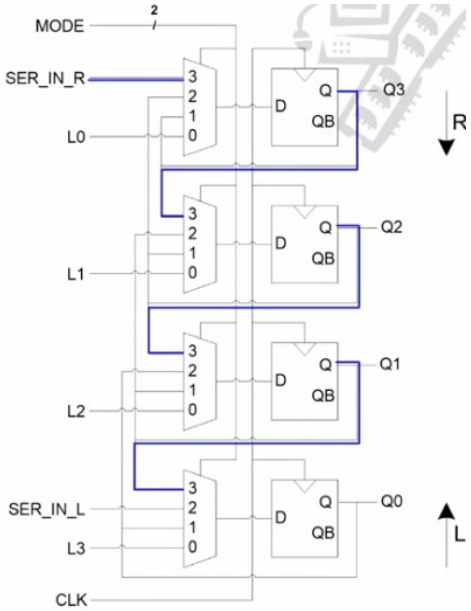
– Hold
  - MODE = 1 (01)



– Shift Left
  - MODE = 2 (10)



– Shift Right
  - MODE = 3 (11)



Using Shift Registers as Counters

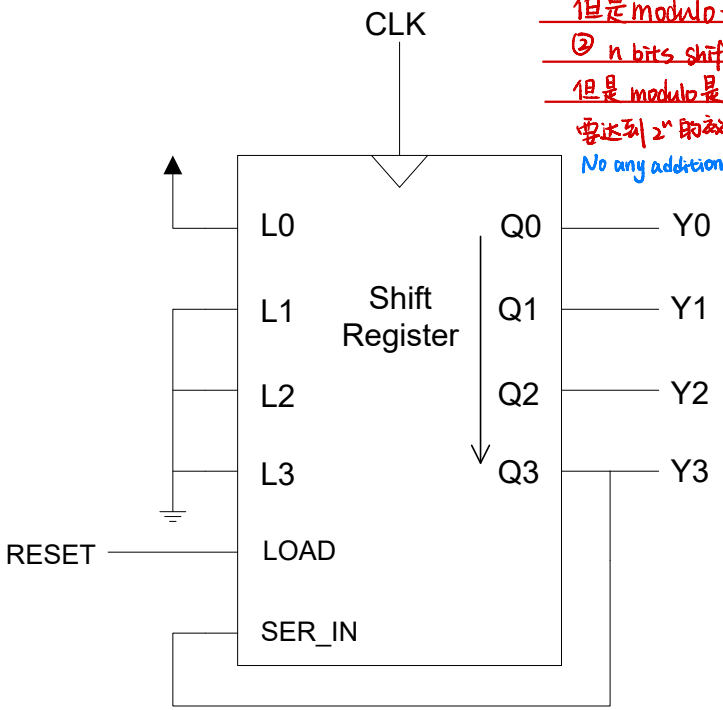
- *n*-bit **ring counter**
  - Implemented with an *n*-bit shift register with the serial out bit fed into the serial in input
  - Counts through sequence of *n* one-hot encodings

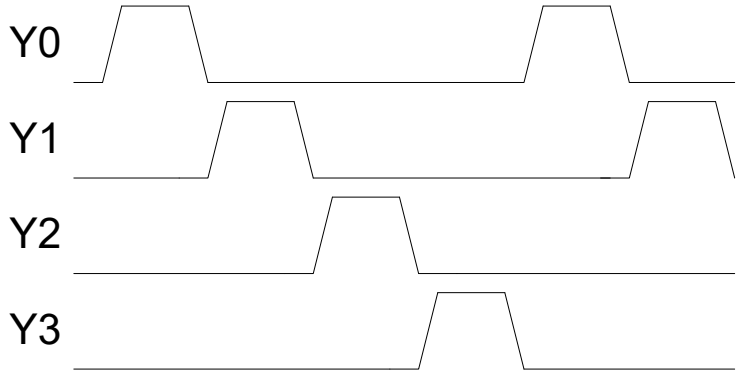
since there are 4 flip flops, So there are 16 states.
However, we only use 4 of them to count.

RESET	Y0	Y1	Y2	Y3
1	x	x	x	x
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1
0	1	0	0	0

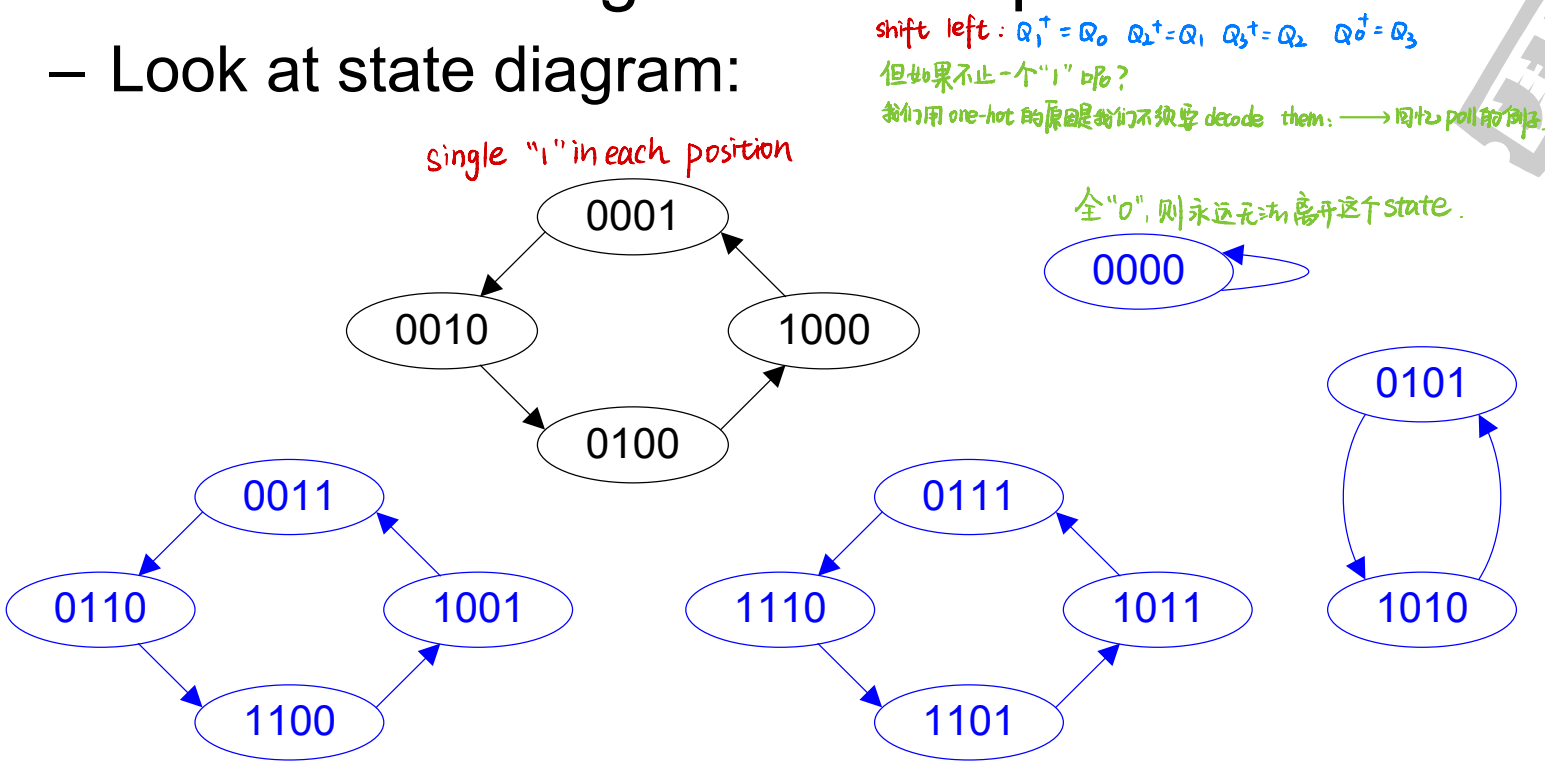
从 table 上看是从左到右.
但是因为 Y_3 是 MSB.
$Y_3\ Y_2\ Y_1\ Y_0$
其实是从右往左.
No extra input here

对比:
① n bits parallel counter logic 复杂
但是 modulo 是 2^n xor & and gate combinational logic.
② n bits shift register logic 简单.
但是 modulo 是 n.
要达到 2^n 的效果,就需要更多的 flip flops.
No any additional gate. it just flip-flops chained together.



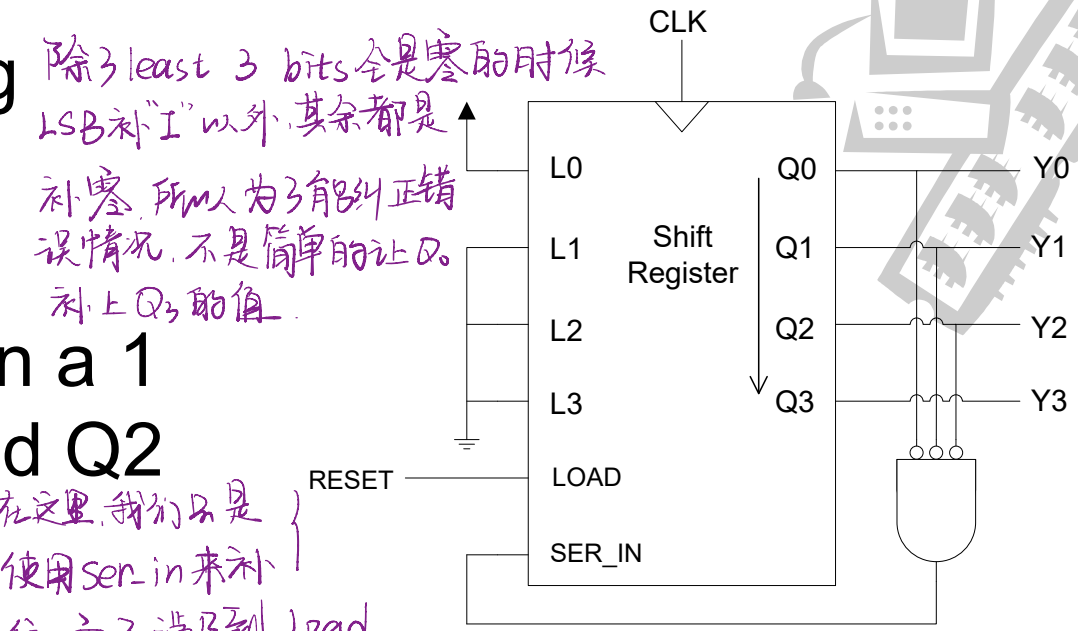


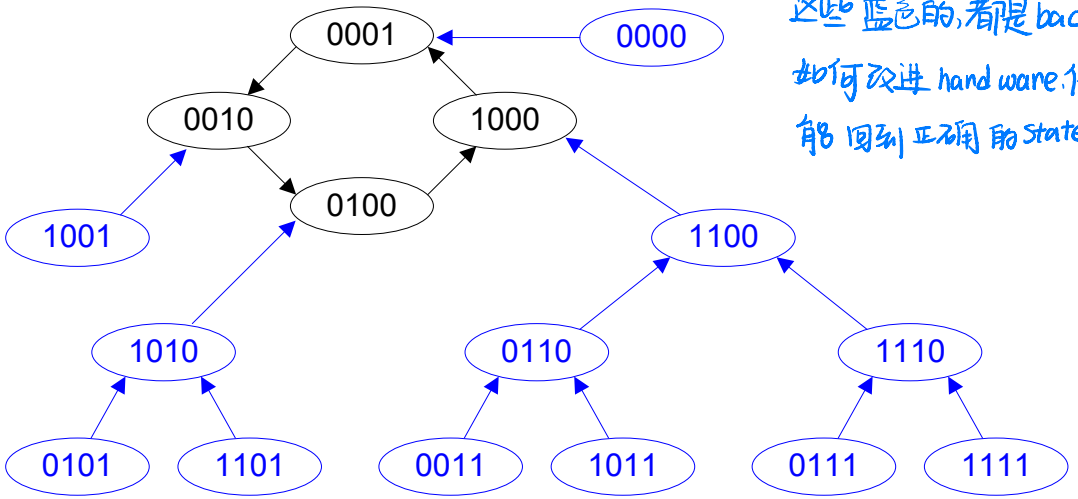
- Problems with ring counter implementation?
  - Look at state diagram:



  - What if noise causes a bit to flip?
    - Counter will never return to its normal sequence

- How to make ring counter robust to noise?
- Idea: Only shift in a 1 when Q0, Q1, and Q2 are equal to 0



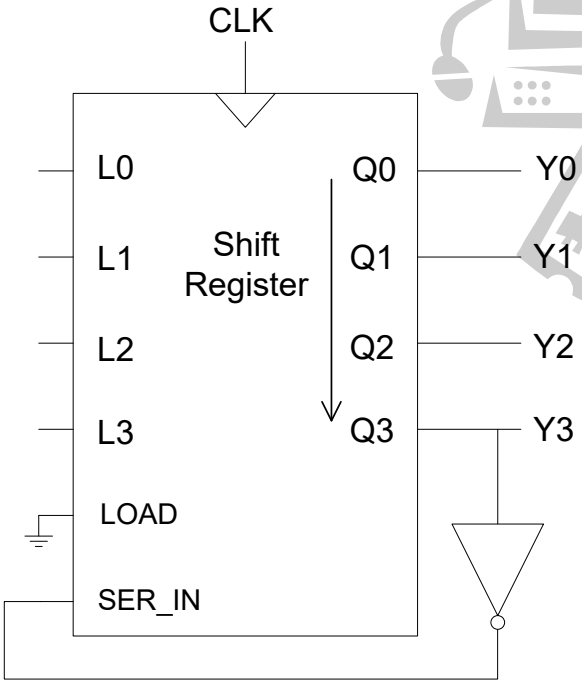


All invalid states will eventually return to the counting sequence!

Q: How many state do Johnson counter has? → How many states can exist.

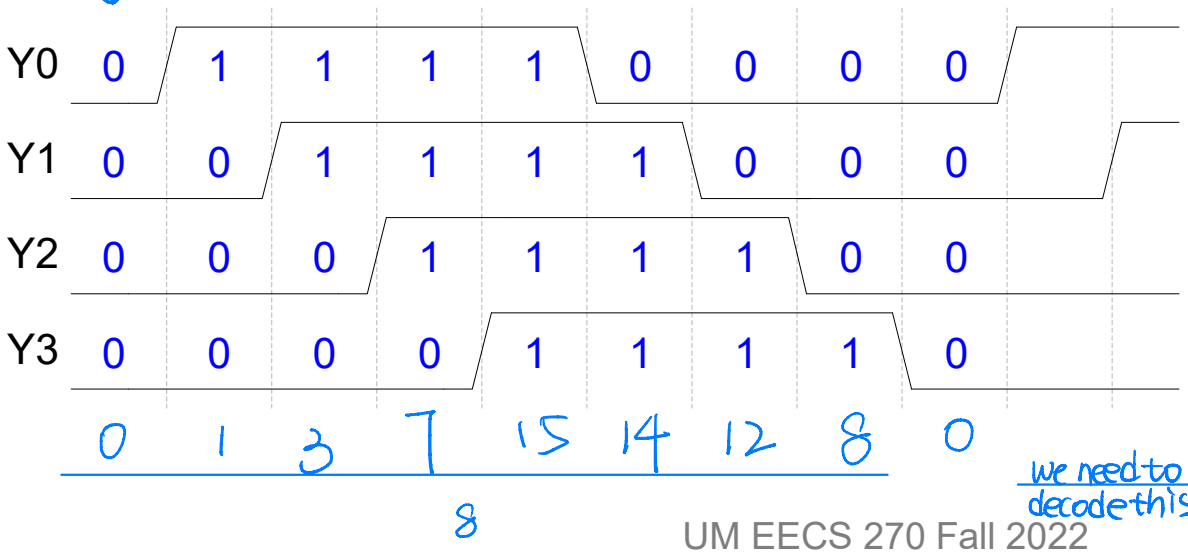
该问题从 flip flop 的数量出发 → 4个ff 所以 16 states. → 我们只想要8个.

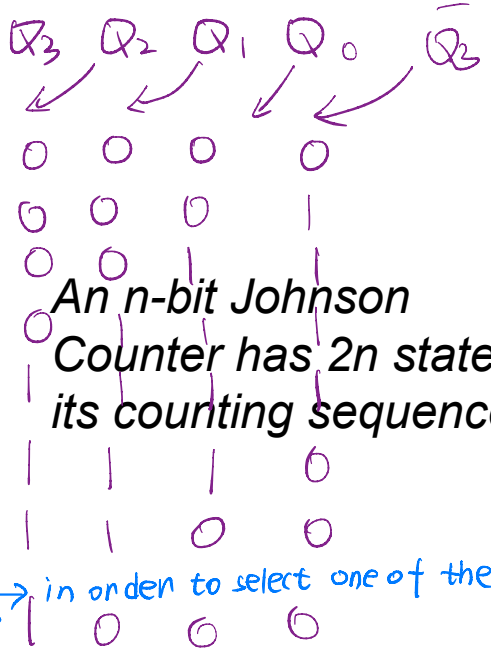
- *n*-bit **Johnson counter** (AKA twist tale counter)
  - Implemented with an *n*-bit shift register with the complement of the serial out bit fed into the serial in input



$Q_1^+ = Q_0 \quad Q_2^+ = Q_1 \quad Q_3^+ = Q_2 \quad Q_0^+ = Q_3'$

Initialize

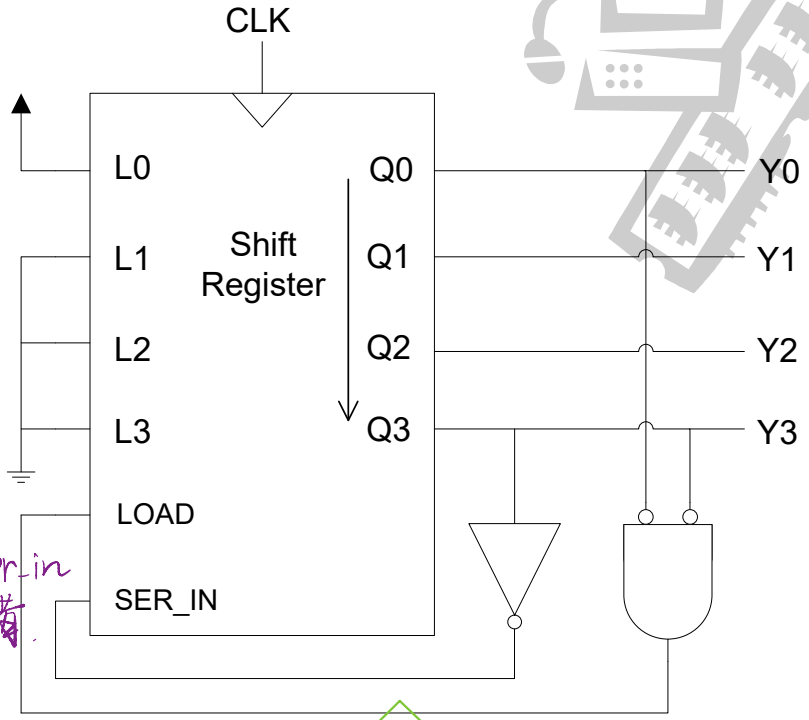




An n-bit Johnson Counter has 2n states in its counting sequence

we need to decode this → in order to select one of the 8 things

- Johnson counters have the same robustness problems as ring counters
- To fix:
  - Every invalid state will reach a state of the form 0xx0
  - When 0xx0 is detected, load 0001 to return to the normal sequence!



在这里我们不改变原有的 Ser_in 逻辑，而是使用 Load 来纠错。

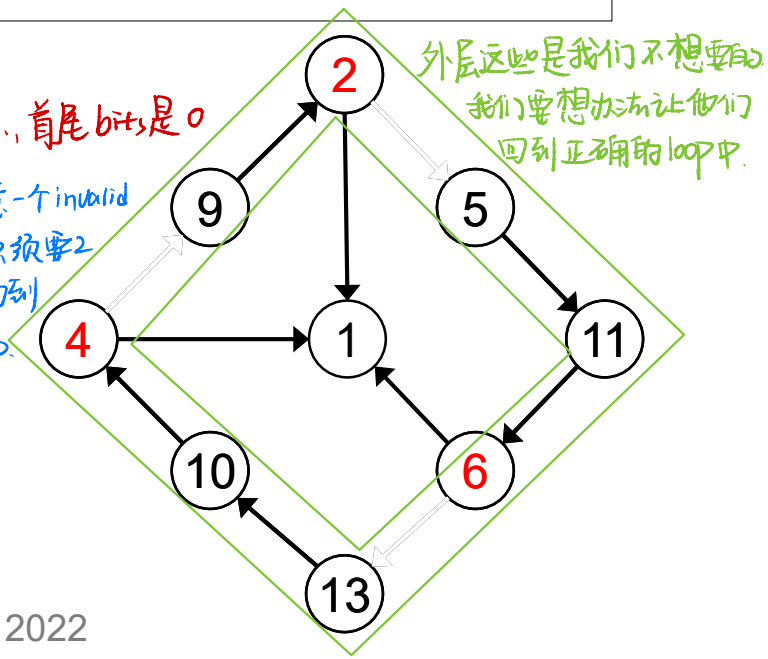
8 gates we want

8 gates we don't want

	Valid States				Invalid States				
	Y3	Y2	Y1	Y0	Y3	Y2	Y1	Y0	
0	0	0	0	0	0	0	1	0	2
1	0	0	0	1	0	1	0	1	
3	0	0	1	1	1	0	1	1	
7	0	1	1	1	0	1	1	0	6
15	1	1	1	1	1	1	0	1	
14	1	1	1	0	1	0	1	0	
12	1	1	0	0	0	1	0	0	4
8	1	0	0	0	1	0	0	1	
	0	0	0	0					

共同特点，首尾 bits 是 0

则，在任意一个 invalid state，最多只须要 2 步就能回到正确的 loop.



外层这些是我们不想要的，我们要想办法让他们回到正确的 loop 中.